CS4375 Operating Systems Concepts
Memory Management Exercise

Assume a 64bit little-endian system with 64bit wide virtual addresses that are first reduced to 48bit effective virtual addresses and then, through 3 levels of page tables to 64bit wide physical addresses. The system uses 4096 byte wide ($2^{12} = 4096$) pages. The MMU first checks that the 17 most significant bits in a virtual address are either all zero or all one. If this condition is not satified, an interrupt is generated by the MMU and a page fault occurs. Otherwise, the MMU continues with the 48 lower bits of the virtual address. The most significant bit of this 48 bit address reflects the bits that have been discarded. The 48 bit address is split into 4 parts:

- An index for a first level of page tables. This first table has 1024 entries.
- A second index for a second level of page tables. This second table has 1024 entries.
- A third index for a third level of page tables. This third table is larger with $65536 = 2^{16}$ entries.
- A fourth part that corresponds to the physical offset inside a page.

Each entry of the tables is only 64bit wide, as the tables use a compressed format: the entry contains the next table's or page frame's address in the 48 least significant bits and it contains 16 bits of flags in the most significant bits. The machine reads a 64 bit entry from a table, analyzes the flags part and then forces the 16 most significant flag bits to all zeros, obtaining a 48bit intermediate address. That 48bit intermediate address is then sign-extended to a 64bit address: if the most significant bit of the 48bit intermediate address is a zero, 16 zeros are prepended; if the most significant of the 48bit intermediate address is a one, 16 ones are prepended to form a 64bit address.

The most significant bit of the 16 bit flags part indicates if the next table or the page is mapped in. If the bit is set to 1, the mapping is valid, otherwise a page fault occurs.

Below you see a table with an extract of the system's physical memory[1]. Use the memory extract and the base address `0x00000000cafe9870` of the first table to translate the virtual addresses

- `0xbabe12340000affe`
- `0xffff8765affecafe`
- `0xffffcafebeefdead`

to physical addresses. If you cannot perform this translation because a page fault occurs, indicate this happening. In your answer, detail each step of the translation; do not just give the final translation result.

Remember that the system is little-endian; this means given an address, e.g. `0x0000000012345600`, you can find the least significant byte `0x88` of the (example) 64bit value `0xaabbccddeeff9988` at the address `0x0000000012345600`, the next byte `0x99` at `0x0000000012345601` and the most significant byte `0xaa` at `0x0000000012345607`.

[1]Everything that starts in `0x` in this exercise is notated in hexadecimal (base 16). Like always, in this notation, the most significant digit is to the left, the least significant one to the right.

Address (64bit)	*Content (64bit)*							
	@+0	*@+1*	*@+2*	*@+3*	*@+4*	*@+5*	*@+6*	*@+7*
0x00000000cafea950	0x1f	0x2a	0x34	0x42	0xea	0xde	0xca	0xfe
0x00000000cafea958	0x70	0xb8	0xfe	0xca	0x00	0x00	0x00	0x80
0x00000000cafea960	0x00	0x00	0xbb	0xde	0x24	0x34	0x10	0xa3
0x00000000cafea968	0x22	0x00	0x11	0x3f	0x3a	0x42	0x19	0xca
0x00000000cafea970	0x2a	0x20	0xbe	0xbe	0x02	0x30	0x23	0x17
⋮	⋮							
0x00000000cafeb1c8	0xf0	0xd8	0xfe	0xca	0x00	0x00	0xff	0xff
0x00000000cafeb1d0	0x00	0xc0	0xaa	0xf7	0x16	0x42	0x00	0x2b
0x00000000cafeb1d8	0x00	0xb0	0xad	0xde	0x17	0x42	0xff	0x2a
0x00000000cafeb1e0	0x00	0xb0	0xad	0xdd	0x18	0x43	0x0f	0x07
⋮	⋮							
0x00000000cafecb20	0x00	0xc0	0xaa	0xf7	0x16	0x42	0x00	0x2b
0x00000000cafecb28	0x00	0xb0	0xad	0xde	0x17	0x42	0xff	0x2a
0x00000000cafecb30	0x00	0xb0	0xad	0xdd	0x18	0x43	0x0f	0x07
0x00000000cafecb38	0xb8	0x80	0xde	0xad	0xca	0xfe	0xaf	0xfe
0x00000000cafecb40	0x70	0x98	0xfe	0xca	0x00	0x00	0xff	0x7f
0x00000000cafecb48	0x00	0xfe	0xfa	0xad	0xde	0x02	0x00	0x01
⋮	⋮							
0x00000000cafef838	0x20	0x44	0x70	0x7f	0x78	0x00	0x08	0x1a
0x00000000cafef840	0x00	0x00	0xbe	0xbb	0x8e	0x0f	0x13	0xff
0x00000000cafef848	0xf8	0xf8	0xfe	0xca	0x00	0x00	0x7e	0x87
0x00000000cafef850	0x08	0x84	0x04	0x0f	0xff	0x10	0x02	0x70
0x00000000cafef858	0x00	0x90	0xbe	0xba	0x8d	0x0f	0x15	0x4b
0x00000000cafef860	0x00	0x90	0xef	0xbe	0x97	0xaf	0x88	0x3a
⋮	⋮							
0x00000000cb0670c0	0x00	0x00	0xbb	0xde	0x24	0x34	0x10	0xa3
0x00000000cb0670c8	0x70	0xb8	0xfe	0xca	0x00	0x00	0x00	0x80
0x00000000cb0670d0	0x2a	0x20	0xbe	0xbe	0x02	0x30	0x23	0x17
0x00000000cb0670d8	0x00	0x1a	0x1a	0x00	0xa0	0x20	0x00	0xff
0x00000000cb0670e0	0x00	0xb0	0xbe	0xba	0xef	0xbe	0x00	0xff
⋮	⋮							
0x00000000cb06f8f8	0x20	0x44	0x70	0x7f	0x78	0x00	0x08	0x1a
0x00000000cb06f900	0x00	0x00	0xbe	0xbb	0x8e	0x0f	0x13	0xff
0x00000000cb06f908	0xf8	0xf8	0xfe	0xca	0x00	0x00	0x7e	0x87
0x00000000cb06f910	0x08	0x84	0x04	0x0f	0xff	0x10	0x02	0x70
0x00000000cb06f918	0x00	0x90	0xbe	0xba	0x8d	0x0f	0x15	0x4b